上級編

第 46 講 待ち行列

例題

人気ラーメン店の行列シミュレーションプログラムを，以下の条件で作成する。

・ラーメン店の座席数，調理時間やお客さんの来るタイミングなどは以下の通りである。

◦ラーメン店の座席数：８席

◦ １杯のラーメンを作るのにかかる時間：３分

◦ １杯のラーメンを食べるのにかかる時間：10 分

◦したがって，一人の客が座席についてから店を出るまでの合計時間：13 分（調理時間３分＋食事時間 10 分）

◦お客さんは１分間に０人～２人到着する。

◦座席が満席の場合は，行列に並ぶ。

◦座席が空いてから片づけを行い，次の客が座席に座るまで１分を要する。

・以下のような形式で，１分ごとに到着客，行列客，席に座っている客の人数を出力する。

実行結果

```
1 分： 到着客：1   行列客：  0   席に座っている客：  1
2 分： 到着客：1   行列客：  0   席に座っている客：  2
3 分： 到着客：2   行列客：  0   席に座っている客：  4
4 分： 到着客：0   行列客：  0   席に座っている客：  4
5 分： 到着客：0   行列客：  0   席に座っている客：  4
6 分： 到着客：2   行列客：  0   席に座っている客：  6
7 分： 到着客：2   行列客：  0   席に座っている客：  8
8 分： 到着客：0   行列客：  0   席に座っている客：  8
9 分： 到着客：2   行列客：  2   席に座っている客：  8
10 分： 到着客：2   行列客：  4   席に座っている客：  8
```

・seats.count(0) という命令は，配列 seats 内の要素「0」の個数を返すものである。

以下のプログラムの空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
import random

seats = [0, 0, 0, 0, 0, 0, 0, 0] # 座席配列（店を出るまでの残り時間を格納）
seats_num = len(seats) # 座席の数
cook_time = 3 # 調理時間
eat_time = 10 # 食べる時間
total_time = cook_time + eat_time # 調理時間 + 食べる時間
waiting = 0 # 行列人数
# 30分間のシミュレーション
for minute in range(30):
  arrival = random.randint(0, 2)
  waiting = waiting + arrival
  # 各席の状態更新
  for i in range(seats_num):
    if seats[i] > 0:
      seats[i] = □
    elif waiting > 0:
      seats[i] = total_time
      waiting = waiting - 1
  print(minute + 1, '分：到着客:', arrival, '  行列客：', waiting, '  席に座っている客：', seats_num - seats.count(0))
```

① seats[i] + 1

② seats[i] - 1

③ seats[i + 1]

④ seats[i] + 1

上級編　第46講　確認問題

以下の条件で，一方通行の道路における交通渋滞シミュレーションプログラムを作成する。

- この一方通行の道路では，渋滞が頻繁に発生する。
- この問題に対処するために，横断歩道の信号のタイミングを調整することで渋滞を緩和できるかシミュレーションすることにした。
- 道路の現状と横断歩道信号のタイミングに関する調査から，次の情報が得られた。
  - 青信号の間，10秒間に最大10台の車が道路を通過できる。
  - 信号は60秒間の青信号と30秒間の赤信号が交互に変わる。
  - 10秒間に道路に到着する車の台数は，ランダムに5台から10台の間で変動する。
- max関数は，引数の中で一番大きいものを返す関数である。

以下のプログラムの空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
import random

totaltime = 18   # 観測時間： 単位10秒（180秒なら18）
passtime = 10    # 通過可能台数
signal_b = 6     # 青信号の時間： 単位10秒
signal_r = 3     # 赤信号の時間： 単位10秒
blue_flag = True # 青信号の場合「True」
arrive = 0       # 車の到着台数
wait = 0         # 渋滞台数

print('秒　到着台数　渋滞台数')

for t in range(1, totaltime + 1):
```

```
    arrive = random.randint(5, 10) # 10 秒間に 5 ～ 10 台ランダムに車が到着する
    mod = (t - 1) % (signal_b + signal_r) + 1
    blue_flag = (mod <= signal_b)
    if blue_flag: # 青信号ならば
      wait = max(□, 0)
    else: # 赤信号ならば
      wait = wait + arrive # 通過できない車はすべて渋滞に加算される
    print(t * 10, '¥t', arrive, '¥t', wait, '¥t')
```

実行結果

秒	到着台数	渋滞台数
10	5	0
20	9	0
30	9	0
40	8	0
50	9	0
60	10	0
70	6	6
80	9	15
90	8	23
100	6	19
110	10	19
120	8	17
130	6	13
140	8	11
150	6	7
160	10	17
170	5	22
180	7	29

① wait + arrive - passtime

② wait + arrive

③ wait + arrive + passtime

④ arrive - passtime

上級編　例題　解答

第34講　コインの枚数

例題　解答：④

第35講　駐車料金の判定

例題　解答：④

第36講　割引のある料金

例題　解答：④

第37講　テストの集計

例題　解答：①

第38講　じゃんけん

例題　解答：①

第39講　サイコロの出目

例題　解答：③

第40講　中央値の判定

例題　解答：①

第41講　素数判定

例題　解答：③

第42講　素因数分解

例題　解答：②

第43講　倍数の判定・FizzBuzz

例題　解答：④

第44講　バブルソート・選択ソート

例題　解答：③

第45講　グラフ理論

例題　解答：②

第46講　待ち行列

例題　解答：②

第47講　パリティチェック

例題　解答：①

第48講　すごろくゲーム

例題　解答：④

上級編　第 46 講　確認問題　解答

正解：①

解説：

if 文の条件式に True か False のどちらかを持つ bool 型を指定すると，その値が True のときに if 文内の処理に入り，False のときは else 文内の処理に入る。つまり，17 行目の `if blue_flag:` では，bool 型の `blue_flag` が True のときは if 文内の処理に入り，False のときは else 文内の処理に入る。

次に if 文内，空欄のある 18 行目を見ていく。変数 wait には，その青信号の時間内に通過しきれなかった車の台数を代入すればよい。

よって正解は① `wait + arrive - passtime` である。

なお，青信号の時間内にすべての車が通過できた場合は，`wait + arrive - passtime` が 0 以下となるため，max 関数によって，もう一つの引数である 0 が返され，変数 wait に代入される。

```
import random

totaltime = 18   # 観測時間：単位 10 秒（180 秒なら 18）
passtime = 10    # 通過可能台数
signal_b = 6     # 青信号の時間：単位 10 秒
signal_r = 3     # 赤信号の時間：単位 10 秒
blue_flag = True # 青信号の場合「True」
arrive = 0       # 車の到着台数
wait = 0         # 渋滞台数

print('秒　到着台数　渋滞台数')

for t in range(1, totaltime + 1):
  arrive = random.randint(5, 10) # 10 秒間に 5 ～ 10 台ランダムに車が到着する
  mod = (t - 1) % (signal_b + signal_r) + 1
  blue_flag = (mod <= signal_b)
  if blue_flag: # 青信号ならば
    wait = max(wait + arrive - passtime, 0)
  else: # 赤信号ならば
    wait = wait + arrive # 通過できない車はすべて渋滞に加算される
  print(t * 10, '¥t', arrive, '¥t', wait, '¥t')
```

実行結果

```
秒　到着台数　渋滞台数
10      5       0
20      9       0
30      9       0
40      8       0
50      9       0
60      10      0
70      6       6
80      9       15
90      8       23
100     6       19
110     10      19
120     8       17
130     6       13
140     8       11
150     6       7
160     10      17
170     5       22
180     7       29
```